Lecture 7: User-Defined Functions*

*Adapted from Chapter 3 of *Think Python: How to Think Like a Computer Scientist, Second Edition*

1. User-defined Functions
2. Where to Place Function Definitions
3. Parameters and Arguments
4. Functions that return results
5. Local and Global Scope
6. Python f-string calling a function

Function - A named sequence of statements that performs some useful operation. Functions may or may not take arguments and may or may not produce a result.

1. User-defined Functions

- We have used Python built-in functions: print(), input(), str(), and int(). Now we will define our own functions.
- *Function definition:* A statement that creates a new function, specifying its name, parameters and the statements it contains.
- Why Use Functions
  - Reusable code - If you use a function in lots of places and have to change it, you only have to edit it in one place (easier to maintain)
  - Clarity - Clearly separated components are easier to read.

Example:

```
def fav_quote():
    print("Do not judge me by my successes, ")
    print("judge me by how many times I fell down ")
    print("and got back up again. ")
    print("- Nelson Mandela")
```

- the `def` keyword indicates that this is a function definition and the name of the function will be `fav_quote()`.
- The empty parentheses indicates that this function doesn't take any arguments.
- Rules for function names are the same as for variable names: letters, numbers and underscore are legal, but the first character can't be a number. You can't use a keyword as the name of a function, and you should avoid having a variable and a function with the same name.
- The first line of the function definition (**header**) must end with a colon.
- The **body** has to be indented (by convention four spaces). The body can contain any number of statements.

function call: A statement that runs a function. It consists of the function name followed by any argument list in parentheses.

```
fav_quote()
```

Calling the new function will produce:

```
Do not judge me by my successes,
judge me by how many times I fell down
and got back up again.
- Nelson Mandela
```

Once you have defined a function, you can call it from inside another function. E.g.,

```
def repeat_quote():
   fav_quote()
   fav_quote()
```

And then a call to `repeat_quote()` will produce:

```
Do not judge me by my successes,
judge me by how many times I fell down
and got back up again.
- Nelson Mandela
Do not judge me by my successes,
judge me by how many times I fell down
and got back up again.
- Nelson Mandela
```

The whole program looks like this:

```
1. def fav_quote():
2.     print("Do not judge me by my successes, ")
3.     print("judge me by how many times I fell down ")
4.     print("and got back up again.")
5.     print("- Nelson Mandela")

6. def repeat_quote():
7.     fav_quote()
8.     fav_quote()

9. repeat_quote()
```

This program contains two function definitions: fav_quote and repeat_quote.

Flow of execution - The Python interpreter reads a program one line at a time from

top to bottom. You need to define a function before its first use (e.g., before the function is called) to ensure that the interpreter knows where the function is located.

- Note the location of the function is known but the code inside the function definition is not run until you call the function.
- **Self-check:** What would the flow of execution be for the above example?

2. Where to Place Function Definitions

A function's def statement must come *before* you call the function to prevent an error:

```
res = return_sum(4,5)   # this will cause an error

def return_sum(x,y):
    c = x + y
    return c
```

3. Parameters and Arguments

- *argument:* A value provided to a function when the function is called. This value is assigned to the corresponding parameter in the function.
- *parameter:* A name used inside a function to refer to the value passed as an argument.

Some functions require arguments. Inside the function, the arguments are assigned to variables called **parameters**. Function that takes an argument:

```
def print_twice(message):
    print(message)
    print(message)
```

This function assigns the argument to a parameter named *message*. When the function is called, it prints the value of the parameter twice. This function works with any value that can be printed:

```
print_twice('Hello')      #Hello
                          #Hello

print_twice(42)           #42
                          #42

greeting = 'Hello'  #variable to use as argument
print_twice(greeting)     #Hello
                          #Hello
```

The name of the variable we pass as an argument (e.g., greeting) is replaced by the parameter name *message* in the *print_twice* function.

4. Functions that return results

- Void functions don't have a return value but might display something on the screen or have some other effect. Some functions return results.
- When you call a function that returns a result you would normally assign the result to a variable or use it as part of an expression otherwise the return value is lost:

```
def return_sum(x,y):
    c = x + y
    return c

res = return_sum(4,5)
```

5. Self-check questions:

1) Parameters and arguments - Which is the correct definition?

a) _____: A name inside a function to refer to the value passed to it.

b) _____: A value provided to a function when the function is called.

2) Write function *test_range()* to check if a number is in the range 1 to 10. Fill in the blanks 1, 2 and 3:

```
(1)___ test_range( (2)___ ):
        if n in range(1,11):
            print("in range")
        else :
            print("outside range.")

(3)_____ (10)
```

3) Write a function that returns the maximum of two numbers.

6. Local and Global Scope

Python has two scopes: **local** and **global**. Local variables are those defined within a function, and are available only within that function. Global variables are those available throughout the program.

Variables created inside a function are **local.** E.g., function cat takes two arguments, concatenates them, and prints the result. When function `cat` terminates, the variable `join` is destroyed. Parameters are also local. E.g., outside function `cat`, there is no such thing as `part1` or `part2`.

```
def cat(part1, part2):
    join = part1 + part2
```

```
    print(join)

line1 = 'Programming '
line2 = 'Principles.'
cat(line1, line2)      # produces ‘Programming
Principles.’
```

A variable that exists in the global scope is a *global variable*. Local and global variables can have the same name, but are different variables with different scopes.

```
def b():
    a = 99          # Creates a local variable named a
    print(a)

a = 42              # Creates a global variable named a
print(a)
b()
print(a)
```

What is the output?

Python f-string calling a function

We can also call functions within f-strings:

```
def mymax(x, y):
    if x > y:
        return x
    return y

a = 3
b = 4
print(f'Max of {a} and {b} is {mymax(a, b)}')
```

The example calls a custom function in the f-string. Output:

```
Max of 3 and 4 is 4
```